Amendments to Claims

This listing of claims will replace all prior versions and listings of claims in the application:

Listing of Claims

1. - 9. (canceled)

10. (currently amended) A method of deadlock management in a multi-thread, parallel processing data management system having ports for sending and receiving data tokens comprising:

allocating at least one thread to a first process and at least one thread to a second process, wherein the first and second processes are connected through a queue via sending and receiving ports;

~~determining~~ using one thread to detect if one or more of said threads are blocked;

if a thread is determined blocked, determining if the blocked thread is sending data or receiving data, wherein a receiving port of said blocked thread blocks if a data token is unavailable and a sending port of said blocked thread blocks when a queue limit is reached; and

determining if a deadlock exists using said block detecting thread by building a wait graph of said one or more blocked threads and determining if the graph is cyclic, wherein if said graph is cyclic, said graph is waiting on itself, indicating a deadlock exists.

11. (original) The method of claim 10, blocking a receiving port when a data token is not available.

12. (original) The method of claim 10, blocking a sending port when a limit on the number of data tokens in the queue is reached.

13. (original) The method of claim 10, including building a wait graph with said blocked threads and traversing said wait graph to determine if it is cyclic.

14. (previously presented) The method of claim 10, if a deadlock is detected, correcting the deadlock by allowing the queue limit of the number of data tokens on a first queue to increase.

15. (original) The method of claim 14, wherein the limit of a queue associated with a sending port is allowed to increase.

16. (previously presented) The method of claim 14, wherein a queue limit on the number of data tokens of a second queue is decreased while said limit of said first queue is increasing.

17. - 20. (canceled)

21. (currently amended) A method for executing a dataflow application comprising:

providing a dataflow application comprising a plurality of map components and data ports, some of said map components being linked between data ports and some map components comprising one or more composite components having a plurality of processes, wherein at least some of said linked data ports being linked by a queue;

allocating a processing thread to a respective map component;

executing multiple processing threads in parallel with each map component on a separate processing thread;

~~detecting~~ using a thread to detect if a deadlock condition does or will exist for one or more of said processing threads by building a wait graph of several thread states and determining if the wait graph is circular; and

correcting a deadlock for a deadlocked processing thread by allowing a first queue linking data ports to exceed a queue limit.

22. (previously presented) The method of claim 21, wherein the correcting step includes choosing a thread that waits as a producer if a circular wait graph is detected.

23. (previously presented) The method of claim 21, wherein if the detecting step determines a wait graph is circular, the correcting step including analyzing queues other than said first queue in the wait graph for token batch reduction.

24. (previously presented) The method of claim 21 wherein if the detecting step determines a wait graph is circular, the correcting step including the substep of reducing said queue limit in one or more queues other than said first queue in the wait graph.

25. (new) A method for executing a dataflow application in a multi-thread processing system comprising:

a) providing a dataflow application comprising a plurality of map components and data ports, a number of map components being linked between data ports using queues and some map components comprising composite components having a plurality of processes;

b) allocating a processing thread to each composite map component including allocating a thread for deadlock detection;

c) executing each composite map component on a separate thread; and

d) determining if a deadlock exists using said deadlock detection thread to monitor queues, including building a wait graph and determining if the graph is cyclic, and if a deadlock exists, allowing a queue limit to increase.

26. (new) The method of claim 25, if a deadlock is detected, correcting the deadlock by allowing the queue limit of the number of data tokens on a first queue to increase.

27. (new) The method of claim 26, wherein a queue limit on the number of data tokens of a second queue is decreased while said limit of said first queue is increasing.

28. (new) The method of claim 25, including transporting data tokens among map components on said queues.

29. (new) The method of claim 28, batching the data tokens to regulate the length of time a map component may execute without synchronization.

30. (new) The method of claim 25, including ports associated with each map component for representing and transporting multi-state null value tokens.

31. (new) The method of claim 30, the null value tokens including an error null.

32. (new) The method of claim 1, said block detection thread monitors one or more of the data queues.

33. (new) The method of claim 32, said block detection thread does not require communication from other threads to determine if a thread is blocked.